Title

The Variable Markov Oracle: Algorithms for Human Gesture Applications

Journal

IEEE Multimedia, 22(4)

ISSN

1070-986X

Authors

Wang, CI

Dubnov, S

Publication Date

2015-10-01

DOI

10.1109/MMUL.2015.76

Peer reviewed

The Variable Markov Oracle: Algorithms for Human Gesture Applications

Cheng-i Wang and Shlomo Dubnov
University of California, San Diego

The Variable Markov Oracle (VMO) data structure for multivariate time-series indexing can identify repetitive fragments and find sequential similarities between observations to support gesture "query by content" and gesture following. Experiments reveal state-of-the-art performance.

Over the past decade, the growing computational power and increasing storage capabilities of personal devices have created the need for tools that let users efficiently, directly, and intuitively interact with multimedia content. Time-series data—such as audio files, videos, and sensor signals—constitutes a major part of multimedia content, but enabling humans to interact with such content requires tackling several problems, including query-by-content, clustering, and classification problems.[1]

Here, we deal with the query-by-content problem in temporal data, addressing both query by content and gesture following. (For more information, see the related sidebar.) Most of the research in this field can be categorized into one of two methods: *model-based* or *sequence-matching*.

Model-based approaches, such as hidden Markov models (HMM),[2] are common in speech, audio, and video-object recognition. An example of a sequence-matching approach is the Guidage system for query by audio.[3] Guidage uses a structure called Audio Oracle (AO) to represent an audio signal in a suffix structure.[4] Guidage can efficiently retrieve subclips by traversing the AO structure in a forward direction, according to an input query. For symbolic subsequence matching, one function absent from AO is a solution for traversing the suffix links. AO is the signal extension of Factor Oracle (FO), which is a suffix automaton for symbolic sequences.

In this article, we propose a new data structure, called the Variable Markov Oracle (VMO), which extends AO and FO by combining the strengths of each. Our work here expands our previous results[5] in both theoretical and practical aspects, exploring the online clustering implication of VMO and gesture following, respectively.

The problem of gesture following stems from the development of interactive art and performance applications.[6] In gesture-following scenarios, the focus is on providing continuous updates of time progression, rather than identifying one correct answer as in query-by-content. Frédéric Bevilacqua has proposed a variant of HMM designed for gesture following,[7] in which the transition probabilities between hidden states, representing time progression, are fixed to allow for efficient real-time algorithms. We propose gesture-following algorithms based on VMO, exploiting its indexing structure for increased efficiency.

The Variable Markov Oracle

As noted, VMO is the successor of FO and AO. FO was originally proposed[8] for biosequence pattern matching and later extended to generating symbolic music sequences.[9] It is a compressed suffix tree (suffix automaton) for the fast retrieval of repeated substrings (factors) and patterns (repeated suffixes) of a symbolic sequence, $\boldsymbol{Q}$.

FO could be constructed in linear time and space, which is shown to be more appropriate in real-time applications than two other variable-order models based on incremental parsing and probabilistic suffix trees.[9] For a sequence of symbol $\boldsymbol{Q} = q_1, q_2, \ldots, q_t, \ldots, q_T$, FO is constructed with T states, and each symbol q_t is associated with a state. Two kinds of links—forward and suffix links—are created during FO construction.

There are two types of forward links in an oracle structure:

- an internal link that is a pointer from state $t - 1$ to t (labeled by the symbol q_t), denoted by $\delta(t-1, \quad q_t) = t$; and

1070-986X/15/$31.00 © 2015 IEEE

Published by the IEEE Computer Society

Query by Content and Gesture Following

There are two main differences between query by content and gesture following. First, you need to decide whether the reporting of the query results will be performed at the end of the query or updated at each new incoming sample of the query time series. Second, query by content asks how closely the query input resembles other entries in the collection, while gesture following considers the time progression of the query time series relative to a stored sequence. Solving both of these problems becomes crucial for helping humans interact with multimedia content, with query by content enabling intuitive browsing of multimedia content, and gesture following providing instantaneous feedback.

Time series query by content has its long history in time series analysis research.[1] Dynamic time warping (DTW) has been the dominant distance metric used in the time series query-by-content task by practitioners but is limited by its quadratic computational complexity[2] and monotonicity conditions. In multimedia research, query by content focuses on using media contents, such as images or audio, as an input to a system in order to retrieve data from a collection of stored samples, according to (or without) user-specified criteria.

As far we know, query by content in multimedia remains an active and relatively unexplored research area. This is in contrast to more standard information retrieval approaches,[3] which focus on devising mid-level features that would allow symbolic searching and matching according to specific tasks. One example studied in recent years in the audio/speech/music domain is query by humming or query by audio.[4]

References

1. P. Esling and C. Agon, "Time-Series Data Mining," *ACM Computing Surveys*, vol. 45, no. 1, 2012, p. 12.
2. M. Cuturi, "Fast Global Alignment Kernels," *Proc. 28th Int'l Conf. Machine Learning* (ICML), 2011, pp. 929–936.
3. M. Worring et al., "Where Is the User in Multimedia Retrieval?" *IEEE MultiMedia*, vol. 19, no. 4, 2012, pp. 6–10.
4. A. Kotsifakos et al., "A Survey of Query-by-Humming Similarity Methods," *Proc. 5th Int'l Conf. Pervasive Technologies Related to Assistive Environments*, 2012, p. 5–8.

- an external link that is a pointer from state t to $t+k$ (labeled by q_{t+k}, where $k > 1$).

An external link $\delta(t, q_{t+k}) = t + k$ is created in FO when $q_{t+1} \neq q_{t+k}, \quad q_t = q_{t+k-1}$, and $\delta(t, q_{t+k}) = \varnothing$. In other words, we create an external forward link when the most recent internal forward link is unseen for the previous occurrence of q_t. The function of the forward links is to provide an efficient way to retrieve any of the factors of $\boldsymbol{Q}$, starting from the beginning of $\boldsymbol{Q}$ and following a unique path formed by forward links.

A suffix link is a backward pointer that points state t to k, where $t > k$. The link does not have a label and is denoted by $\mathbf{sfx}[t] = k$. The condition for when a suffix link is created is

$\mathbf{sfx}[t] = k \iff$ the longest repeated suffix of $\{q_1, \quad q_2, \ldots, q_t\}$ is recognized in k.

Suffix links are used to find repeated suffixes in $\boldsymbol{Q}$. To track the longest repeated suffix at each time index t, the length of the longest repeated suffix at each state t (denoted as `lrs`[t]) is computed by the algorithm proposed Arnaud Lefebvre, Thierry Lecroq, and Joël Alexandre.[10] FO construction could be done incrementally by appending newly appearing symbols to the end of $\boldsymbol{Q}$. (The algorithms for constructing FO are provided elsewhere.[10]) Figure 1 shows an example of the FO structure.

AO is the multivariate time-series extension of FO. The input $\boldsymbol{O}[t]$ is a multivariate time series sampled at discrete times. To extend the domain of FO from symbolic sequences to a time series, such as an audio signal, a threshold θ is introduced to determine if $\boldsymbol{O}[t]$ is similar to states found in $\boldsymbol{O}[1 \ldots t-1]$ by following suffix links. θ is associated with the metric over the feature space given the time series. Two samples, $\boldsymbol{O}[i]$ and $\boldsymbol{O}[j]$, are considered similar if $|\boldsymbol{O}[i] - \boldsymbol{O}[j]| \leq \theta$. The metric should be chosen according to the application domain and features used. For the rest of the article, we use L_2-norm as the metric between feature frames.

Online Construction Algorithm for VMO

As mentioned, AO resembles the suffix structure of FO without symbols attached to states. This fact prevents AO from having efficient navigating algorithms for states linked by suffix links. Consequently, we devised VMO to improve AO by explicitly assigning labels to frames linked by suffix links during AO construction.

The symbols formed by gathering states connected by suffix links have the following properties:

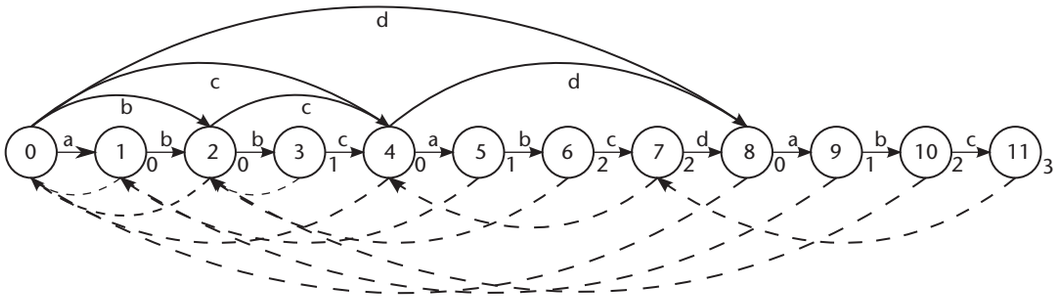

Figure 1. An example of the Factor Oracle (FO) structure. FO of $\boldsymbol{Q}$ *= "abbcabcdabc." The forward links include the labels of each symbol. (Each state has an associated symbol in FO.)*

Require: Time series as $O = O[1]; O[2] \ldots O[T]$

1. Create an oracle P with initial state p_0
2. $\mathbf{sfx}_P$ [0] ⟵ -1, Σ ⟵ ∅, M ⟵ 1
3. **for** t = 1 : T **do**
4. $Oracle(P = p_1 \ldots p_t)$, M ⟵ Add-Frame($Oracle(P = p_1 \ldots p_{t-1})$, O[t], M)
5. **end for**
6. **return** $Oracle(P = p_1 \ldots p_T)$

Figure 2. Algorithm 1—Variable Markov Oracle construction. This algorithm provides online construction of VMO.

- pairwise distances between states connected by suffix links are less than θ;
- the symbolized signal formed by the oracle could be interpreted as a sample from a variable-order Markov model, because the states connected by suffix links share common suffixes with variable length;
- each state is labeled by only one symbol, because each state has only one suffix link; and
- the alphabet size of the assigned symbols is unknown before the construction and is determined by θ.

To explicitly keep track of the linked states and to maintain the online nature of the construction algorithm, the construction of VMO combines FO and AO by treating the sequence of assigned labels $\boldsymbol{Q}$ as the symbolic sequence for FO. Pointers to $\boldsymbol{O}$ are tracked by introducing a list of lists, $\sum = [\sigma_1 \cdots \sigma_m \cdots \sigma_M]$, where M is the number of labels formed and σ_m is a list containing the pointers (frame numbers) for the mth label. In summary, VMO accepts $\boldsymbol{O}$ as input and returns an oracle structure, keeping track of the cluster label sequence $\boldsymbol{Q}$ and also the lists of pointers to $\boldsymbol{O}$. The lists are stored in $\sum$ and indexed by $\boldsymbol{Q}$.

Let $\boldsymbol{O}$ be the incoming signal and t the time index. We use $\boldsymbol{O}[t]$ to represent the newly observed value or vector at t. A forward link from state i to state j, labeled by q, is denoted by $\delta(i, \quad q) = j$. A suffix link from state j to state i is denoted by $\mathbf{sfx}[j] = i$ without labeling. We use $\boldsymbol{Q} = q_1, \ldots, q_T$ to denote the label sequence for labels of observations $\boldsymbol{O} = \boldsymbol{O}[1] \ldots \boldsymbol{O}[T]$. Algorithm 1 (shown in Figure 2) provides VMO a construction.

Algorithm 2 provides the incremental algorithm for an incoming signal (see Figure 3). For each new incoming sample $\boldsymbol{O}[t]$, a new state is constructed with the newly created internal forward link $\delta(t-1, \quad q_t) = t$. The cluster label q_t for $\boldsymbol{O}[t]$ is initialized as null. The `while` loop (lines 5 to 15) in Algorithm 2 is the standard process to assign external forward links and suffix links.[10] Lines 16 to 25 are the newly introduced part of VMO that assigns the cluster label

```
Require: Oracle P = p_1 … p_t, time-series instance O[t + 1] and number of cluster M
    1.  Create a new state t + 1
    2.  q_{t+1} ← 0, sfx_P [t + 1] ← 0
    3.  Create a new transition from t to t + 1, δ(t, q_{t+1}) = t + 1
    4.  k ← sfx_P [t]
    5.  while k > −1 do
    6.     D ← distances between O[t + 1] and O[δ(k, :)]
    7.     if all distances in D are greater than θ then
    8.         δ(k; q_{t+1}) ← t + 1
    9.         k ← sfx_P [k]
    10.    else
    11.        Find the forward link from k that minimizes D
               k′ ← (k; :)[argmin(D)]
    12.        sfx_P[t + 1] ← k′
    13.        break
    14.    end if
    15. end while
    16. if k = −1 then
    17.      sfx_P [t + 1] = 0
    18.      Initialize a new cluster with current frame index σ_{M+1} ← t + 1
    19.      Σ ← [Σ; σ_{M+1}]
    20.      Assign a label to the new cluster, q_{t+1} ← M + 1
    21.      Update number of clusters, M ← M + 1
    22. else
    23.      Assign cluster label based on assigned suffix link q_{t+1} ← q_{k′}
    24.      σ_{q_{k′}} ← [σ_{q_{k′}}; t+1]
    25. end if
    26. reture Oracle P = p_1 … p_{t+1}, M
```

Figure 3. Algorithm 2—the incremental algorithm for an incoming signal. For each new incoming sample $\mathbf{O}[t]$*, a new state is constructed with the newly created internal forward link* $\delta(t-1, q_t) = t$*. Lines 16 to 25 show the newly introduced part of VMO that assigns the cluster label to* q_t*, then appends the pointer of* $\mathbf{O}[t]$ *to* σ_{qt}*.*

to q_t, then appends the pointer of $\boldsymbol{O}[t]$ to σ_{q_t}. In this article, for the algorithms described in pseudocode, we use $X[i]$ when retrieving the item from an array X in its ith location; $[a; b]$ when appending b to the end of a; $X_{i,j}$ when accessing the ith row and jth column of a matrix X; and $X(i, :)$ when retrieving the whole ith row in a matrix X.

Online Clustering

The online construction algorithms build a clustering of frames, storing the clustering in $\sum$. We conjecture how the construction algorithms could be viewed as an online clustering algorithm with Markov constraints. Online clustering refers to incrementally aggregating a datastream into clusters without observing all of the data points in the first place. David Stork, Peter Hart, and Richard Duda proposed the leader-follower online clustering algorithm,[11]

```
Require: A data stream with x the incoming sample
    1. Initialize  η, θ
    2. w_1 ← x
    3. while x is presented do
    4.      j ← argmin_j′ ‖x − w_j′‖ (find nearest cluster)
    5.      if ‖x − w_j′‖ < θ then
    6.          w_j ← w_j + ηx
    7.      else
    8.          add new w ← x
    9.          ‖w‖ ← w/‖w‖  (normalize weight)
    10.     end if
    11. end while
    12. return w_1, w_2, …
```

Figure 4. Algorithm 3—leader-follower clustering.[11] *The clusters are formed dynamically during the observation of a signal.*

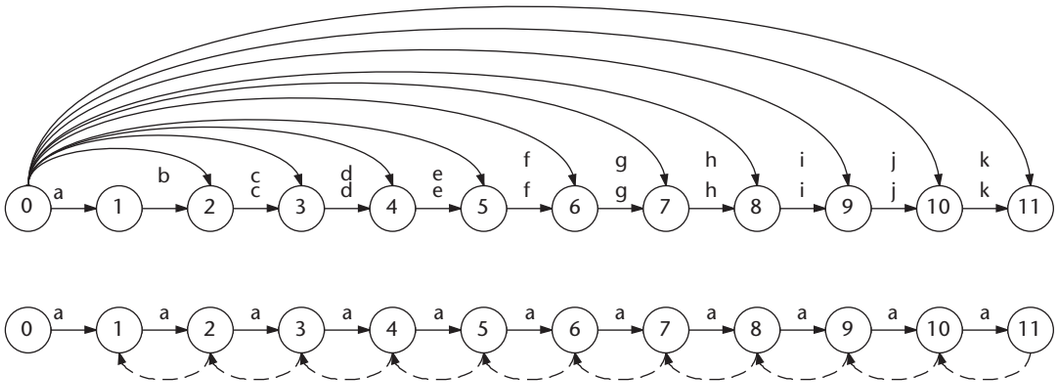

Figure 5. Two oracle structures with extreme values of θ. The characters near each forward link represent the assigned labels. In the top oracle structure, the θ value is 0 or extremely low. In the bottom oracle structure, the θ value is very high. In both cases, the oracles can't capture any type of structure from the time series.

which we provide here as Algorithm 3 (see Figure 4).

Comparing Algorithm 2 to Algorithm 3, we can observe that the suffix-link finding routine in Algorithm 2 is similar to line 4 in Algorithm 3, where a possible cluster label is identified. Then the label-assigning routine in Algorithm 2 is where either a label gets assigned or a new label is formed, which is similar to the If-else part in Algorithm 3. The θ used in both algorithms plays the same role in the sense that it controls the number of clusters created. However, two major differences separate these two algorithms.

First, the suffix-link finding routine in Algorithm 2 places a Markov constraint on the states to be considered as candidate states for the suffix link of the incoming sample. Markov constraint refers to the constraints placed in a constraint satisfaction problem (CSP) to enforce Markov properties in the generated sequence from solving the CSP.[12] We borrow the concept and use it to explain the effect of line 6 in Algorithm 2 where, instead of having Markov constraint cost functions in an optimization scenario of CSP, VMO directly limits only the states sharing the same suffix with the incoming observation to be considered as candidates to which suffix links point. Then variable-order Markov properties are injected because of the shared suffixes.

The second difference is that VMO does not parametrize the clustered frames in terms of centroids or other statistics, because the clusters in VMO are formed by assigning linkages between incoming and existing frames or using criteria related to distance or similarity measures. VMO is thus more like a variant of single linkage hierarchical clustering.

Model Selection via the Information Rate

For an extreme θ value, VMO can assign different symbols to every frame in $\boldsymbol{O}$ if θ is excessively low, or VMO can assign the same symbol to every frame in $\boldsymbol{O}$ if θ is excessively high. In these two cases, VMO can't capture variable-order dependencies in the time series. Figure 5 shows an example of the oracle structure with extreme θ values.

With different θ values, VMO constructs different suffix structures and different symbolized sequences from the signal. To select the one symbolized sequence with the most informative variable-order structure, we use information rate (IR) as the criterion in model selection between different structures generated by different θ values. IR is an information theoretic measure capable of measuring the information content of a time series[4] in terms of the predictability of its source process on the present observation given past ones. VMO uses the same approach as AO[13] to calculate IR. Let $x_1^N = \{x_1, x_2, \ldots, x_N\}$ denote time series x with N observations, where $H(x) = -\sum P(x) \log_2 P(x)$ is the entropy of x and the definition of IR is

$$IR(x_1^{n-1}, x_n) = H(x_n) - H(x_n|x_1^{n-1}). \qquad (1)$$

IR is the mutual information between the present and past observations, which is maximized when there is a balance between

variation and repetition in the symbolized signal. The value of IR could be approximated by replacing the entropy terms in Equation 1 with a complexity measure C associated with a compression algorithm. This complexity measure is the number of bits used to compress x_n independently using the past observations x_1^{n-1}. Consequently,

$$IR(x_1^{n-1}, x_n) \approx C(x_n) - C(x_n|x_1^{n-1}).$$

Compror, a lossless compression algorithm based on FO and the length of the longest repeated suffix link (**lrs**) is provided elsewhere,[6] as is the detailed formulation of combining Compror, AO, and IR.[13]

For VMO, the online generation of code words C is as follows[6]: When a new state is added to an oracle structure at time $(t + 1)$, the suffix link and length of the longest repeated suffix for this state are retrieved. If the resulting suffix link points to state *0,* that means no suffix was found because the distance between the new state and all previous states exceeded the threshold θ. In such a case, the new state must be individually encoded. Otherwise, if a suffix link pointing to a previous location in the state sequence is found, and the length of the longest repeating suffix is smaller than the number of steps passed since the last encoding event, then a complete preceding block of states is encoded in terms of a pair (*length, position*).

In our method, we denote an individual new state as a pair (0, *position*). Let us denote $\boldsymbol{\kappa}[i]$ as the array that contains the states where encoding occurs during the compression pass. An algorithm for computing $\boldsymbol{\kappa}$ is described in Algorithm 4 (see Figure 6), with **lrs** representing an array containing the length of repeated suffixes of the oracle. **lrs** could be obtained during the construction of an oracle. The collection of code pairs $\Phi\boldsymbol{\kappa}$ resulting from Algorithm 4 is passed to the incremental IR algorithm, as described in Algorithm 5 (see Figure 7).

Figure 8 visualizes the sum of IR values versus different θs. When Algorithms 4 and 5 are applied to VMO, the higher θ value creates higher $C(x_n)$ and $C(x_n|x_1^{n-1})$, while lower θ creates lower $C(x_n)$ and $C(x_n|x_1^{n-1})$. Thus IR is maximized with an intermediate θ value when $C(x_n)$ and the negative $C(x_n|x_1^{n-1})$ are balanced. A VMO with a higher IR value captures more of the repeating subclips (such as patterns, motifs, themes, and gestures) than the ones with lower IR values.

```
Require: Array containing the length of repeated
suffixes for every state lrs[t]; t = 1 … T

1. Create an array κ with initialization κ = {1}
2. for t = 1 to T do
3.     if lrs[t + 1] < t − lrs[end] + 1 then
4.         κ ← κ ∪ {t}
5.     end if
6. end for
7. return Vector κ
```

Figure 6. Algorithm 4—compression pass over VMO.* κ[i] *denotes an array that contains the states where encoding occurs during the compression pass. This is an algorithm for computing* κ*, with* **lrs** *representing an array containing the length of repeated suffixes of the oracle.

Matching Algorithms

Let R be the query observation indexed by n, denoted as $\boldsymbol{R} = \boldsymbol{R}[1],\ldots, \boldsymbol{R}[n],\ldots, \boldsymbol{R}[N]$. The matching algorithm provided in Algorithm 6 (see Figure 9) takes R as input and matches it to the target VMO, *Oracle* $(\boldsymbol{Q} = q_1, q_2, \ldots, q_T, \boldsymbol{O} = \boldsymbol{O}[1], \boldsymbol{O}[2], \ldots, \boldsymbol{O}[T])$, constructed by a target time series, $\boldsymbol{O}$. The algorithm returns a cost and a corresponding recombination path. The cost is the reconstruction error between the query and the best match from $\boldsymbol{O}$, given a metric on a frame-by-frame basis. The recombination path corresponds to the sequence of indices that will reconstruct a new sequence from $\boldsymbol{O}$ that best resembles the query.

Query-Matching Algorithm

Our proposed query-matching algorithm is a dynamic programming algorithm. We separate the algorithm into two steps: initialization and decoding. In Algorithm 6, the initialization is in lines 1 through 6. During initialization, the size of the alphabet, M, is obtained from the cardinality of $\sum$. Then, the frame within the mth list that is closest to the first query frame, $\boldsymbol{R}[1]$, is found and stored. After the initialization step, the decoding step (lines 7 through 13 in Algorithm 6) iterates over the rest of the query frames from 2 to N to find M paths, with each path beginning with the state found corresponding to the respective label in the initialization step.

Require: A sequence of codeword pairs Φ = (*length*, *position*), with position information, κ, from compression algorithm, and length information from **lrs**.

1. Compute length T by summing all $\Phi \rightarrow$ *length* values
2. Create vectors $C(x_n)$ and $C\left(x_n \middle| x_1^{n-1}\right)$
3. **for** t = 1 to T **do**
4. $M \leftarrow$ number of symbols up to t by $\sum_{i=1}^{t} (\text{lrs}[i] == 0)$
5. $C(x_t) \leftarrow \log_2(\text{M})$

$$C\left(x_t \middle| x_1^{t-1}\right) \leftarrow \frac{\log_2(\#\text{ of codewords } \Phi \text{ up to } t)}{\Phi \rightarrow \textit{length} \text{ to which state } t \text{ belongs}}$$

6.
7. **end for**
8. **return** Vector $IR = C(x_n) - C\left(x_n \middle| x_1^{n-1}\right)$

Figure 7. Algorithm 5—incremental information rate (IR) from code Φ. The collection of code pairs $\Phi\kappa$ resulting from Algorithm 4 is passed to the incremental IR algorithm.

The proposed query-matching algorithm is similar to the Viterbi decoding algorithm for HMM and max-sum inference algorithm for graphical models[14] in the sense that each update in the decoding step depends only on its neighboring findings. Thus, it's efficient to compute, and there is no need to search over the whole state space. Figure 10 provides a visualization of Algorithm 6 from initialization to decoding for one path among the M paths.

Online Alternative

To fulfill the online requirement for the proposed gesture-following task, we adapted the query-matching algorithm described in Algorithm 6. The online algorithms are provided in Algorithms 7 and 8 (see Figure 11). The online gesture following updates the current "time progression" and "likelihood" on a frame-by-frame basis instead of making one final backward pass at the end of the query time series for the offline version.

In Algorithm 7, M possible paths are initialized according to the number of labels assigned by VMO and the first entry for each path is located according to the distance between the first observation from the input stream to every frame in each label. Algorithm 7 returns a path vector P and a cost vector C and passes them to the online following algorithm. In Algorithm 8, the algorithm is run once to update P and C for each path for each incoming sample, then $P_{arg\,min\,(C)}$ is returned as the found index in target time series **O,** indicating the best match between the incoming time series $\boldsymbol{R}$ and stored time series $\boldsymbol{O}$.

A Probabilistic Interpretation

Here, we specify the analogy between the inference problem in HMM and the query-matching algorithm used by VMO. Given an HMM learned from $\boldsymbol{O} = \boldsymbol{O}[1], \boldsymbol{O}[2],\ldots, \boldsymbol{O}[T]$—specified as initial probabilities π_i of being at state i; transition probabilities a_{ij} for transitioning from state i to state j; the hidden state at time step n denoted by x_n; and emission probabilities $P(R[n]|m)$ for the observed variable $\boldsymbol{R}[n]$ at time step n generated by state m—and given an observed time series $\boldsymbol{R}$, then

$$V_{1,m} = P(\mathbf{R}[1]|m) \cdot \pi_m, \quad \text{and}$$
$$V_{n,m} = \max_{m'} \left(P(\mathbf{R}[1]|m) \cdot a_{m,m'} \cdot V_{n-1,m'}\right)$$

Given VMO indexing $\boldsymbol{O} = \boldsymbol{O}[1], \boldsymbol{O}[2],\ldots, \boldsymbol{O}[T]$, we can replace the terms in the recurrence relations in HMM with attributes in VMO.

First, we treat the clusters of frames $\sigma_1, \ldots, \sigma_M$ stored in $\sum$ as hidden states in an HMM. With $\sum$ available from VMO, initial probability is replaced by an empirical frequency estimation from VMO as

$$\pi_i = \frac{|\sigma_i|}{T},$$

with $|\sigma_i|$ denoting the number of frames stored in σ_i. The transition probability is replaced with

$$a_{ij} = \frac{1\left(\exists \delta(t,j), t \in \sigma_i\right)|\sigma_j|}{\sum_{j'=1}^{M} 1\left(\exists \delta(t,j'), t \in \sigma_i\right)|\sigma_{j'}|},$$

where 1() is an indicator function that returns 1 if the condition enclosed is true, or 0 otherwise. The replacement for the transition probability is an approximation with conditional empirical frequency estimation from transition candidates by forward links. We replace the emission probability with

$$P(\mathbf{R}[n]|m) \propto \exp\left(\frac{-d(\mathbf{R}[n]|m)}{\alpha}\right),$$

where $d(R[n], m) \geq 0$ and $\alpha \geq 0$. Note that α is a scalar controlling the variance, assumed to be 1, and $d(R[n], m)$ is a cost function, which we define as

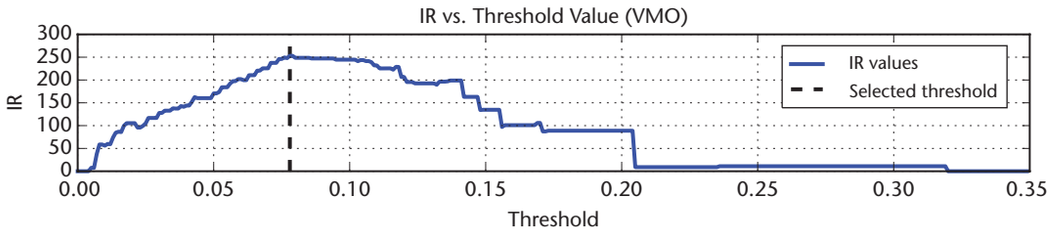

Figure 8. IR values are shown on the vertical axis, while θ are on the horizontal axis. The solid blue curve shows the relationship between IR and θ, and the dashed black line indicates the chosen θ by locating the maximum IR value. Empirically, IR curves exhibit quasi-concave function shapes, so a global maximum can be located.

$$d(\mathbf{R}[n], m) \triangleq \min_{t' \in \sigma_m} \|\mathbf{R}[n] - \mathbf{O}[t']\|.$$

The definition of the cost function refers to finding the closest frames in σ_m to the incoming observation $R[n]$ and returning their distance. If we take `log` on both sides of the recurrence relations and combine the above replacements, the recurrence relations could be rewritten as

$$\begin{aligned}
&\log(V_{1,m}) \\
&= \log\left(exp\left(-\min_{t' \in \sigma_m} \|\mathbf{R}[1] - \mathbf{O}[t']\|\right) \cdot \frac{|\sigma_m|}{T}\right) \\
&= -\min_{t' \in \sigma_m} \|\mathbf{R}[1] - \mathbf{O}[t']\| + \log\left(\frac{|\sigma_m|}{T}\right), \\
&\log(V_{n,m}) \\
&= \max_{m'} \log\left(\left(exp\left(-\min_{t' \in \sigma_m} \|\mathbf{R}[n] - \mathbf{O}[t']\|\right)\right)\right. \\
&\quad \left. \cdot \frac{1\left(\exists \delta(t,m), t \in \sigma_{m'}\right)|\sigma_m|}{\sum_{j=1}^{M} 1\left(\exists \delta(t,j), t \in \sigma_{m'}\right)|\sigma_j|} \cdot V_{n-1,m'}\right) \\
&= \min_{m'}\left(\min_{t' \in \sigma_m} \|\mathbf{R}[n] - \mathbf{O}[t']\|\right. \\
&\quad \left. + \log\left(\frac{1\left(\exists \delta(t,m), t \in \sigma_{m'}\right)|\sigma_m|}{\sum_{j=1}^{M} 1\left(\exists \delta(t,j), t \in \sigma_{m'}\right)|\sigma_j|} \cdot V_{n-1,m'}\right)\right).
\end{aligned} \tag{2}$$

We can then compare this rewritten recursive relation to the query-matching algorithm proposed in Algorithm 6. We first notice that Algorithm 6 not only identifies the hidden state ($\sigma_{m'}$) but also the extracted frame $\boldsymbol{O}[t']$ from that hidden state, which is consistent with the cost function $\min_{t' \in \sigma_m} \|\mathbf{R}[n] - \mathbf{O}[t']\|$ just proposed. The next observation is that the probabilities for transitioning to different σ_m is the number of frames in σ_m proportional to the sum of number of frames possible to transition to from $\sigma_{m'}$. Combining these two observations could establish that the probability distribution for candidate frames before considering the emission probability is actually uniform. Thus, we need to consider only the cost function in Equation 2, which means $\log(V_{1,m})$ and $\log(V_{n,m})$ are actually C_m in Algorithm 6.

Query by Content

The goal for the first part of the experiment is to test the performance of Algorithm 6. We formalize the problem as follows: given a collection of time series data of the same type (such as audio, video, or sensory data) categorized into different categories (or types, genres, or labels) and a query time series not included in the collection, use the query time series to search over the collection and return multiple matches according to the query. The returned matches are deemed correct if they belong to the same category as the query.

For this experiment, we choose the MSRC-12 Kinect gesture dataset,[15] which contains 6,244 annotated instances of both iconic and metaphoric human full-body gestures recorded using XBox Kinect by 30 participants. The sample rate of the 3D joints is 30Hz and each frame of the gesture instance is stored as a 3D 20-joint skeleton. We treat the sequence of skeleton frames as the query time series, with $R \in \mathbb{R}^{3\times 20}$. Figure 12 shows snapshots of example skeleton frames. The length of each gesture sequence is between 13 and 492 frames, and each participant performed 8 to 20 times for each gesture.

The experiment follows the leave-one-out principle. First, all the instances are preprocessed for storage in VMO structures. Then, each time one instance from the collection is set aside as the query, we use the skeleton

```
Require: Target signal in VMO, Oracle(Q = q_1, q_2, … , q_T, O = O[1], O[2], …,
O[T]) and query time series R = R[1], R[2], … ,R[M]
    1. Get the number of clusters, M ← |Σ|
    2. Initialize cost vector C ∈ ℝ^M and path matrix P ∈ ℝ^(M×N).
    3. for m = 1 : M do
    4.       P_{m,1} ← Find the state, t, in the mth list from Σ
                with the least distance, d_{m,1}, to R[1]
    5.       C_m ← d_{m,1}
    6. end for
    7. for n = 2 : N do
    8.       for m = 1 : M do
    9.          P_{m,n} ← Find the state, t, in lists with labels
                    corresponding to forward links from state
                    P_{m,n−1} with the least distance, d_{m,n} to R[n]
   10.          C_m + = d_{m,n}
   11.       end for
   12. end for
   13. return P[argmin(C)], min(C)
```

Figure 9. Algorithm 6—query matching. The algorithm returns a cost and a corresponding recombination path. The cost is the reconstruction error between the query and the best match from ***O****, given a metric on a frame-by-frame basis. The recombination path corresponds to the sequence of indices that will reconstruct a new sequence from* ***O*** *that best resembles the query.*

sequence of the query as $\boldsymbol{R}$ in Algorithm 6 to match the rest of the instances in the collection stored as VMOs. Algorithm 6 returns the 10 entries with the lowest cost, C, for each query. Every instance is used as a query once in this experiment. We choose a smaller subset instead of the full collection due to the large amount of instances. The subset chosen contains the six iconic gestures out of the 12 gestures performed by five randomly chosen participants from the full collection. The total number of instances of the subset is 499. A list of the details of the subset is provided in Table 1.

The results of the leave-one-out query-return experiment are listed in Tables 2 and 3. In Table 2, a retrieved match is considered correct if it belongs to the same category of the query for each of the 10 retrievals of the query; if so, then the precision for one type of gesture is calculated as the "counts of correct matches" divided by the "total number of queries from that type of gesture." We compare the result in Table 2 to the state of the art on reported precision on the MSRC-12 dataset.[16] The precision achieved using the VMO query-matching algorithm (93.1 percent) reached a comparable level (93.6 percent) on the subset chosen in this experiment.

Researchers have proposed a feature called Cov3DJ,[16] in which the covariances between joints are used and a support vector machine (SVM) is trained to recognize testing gestures. SVM with Cov3DJ reaches the state-of-the-art performance, but because Cov3DJ must be calculated over all the data points of each gesture, it is not possible for online applications. We also compare our results to dynamic time warping (DTW), HMM, and experiments[17] on the MSRC-12 dataset.

DTW and HMM are both baseline approaches for time-series query-retrieval experiments. Mallinali Ramírez-Corona, Miguel Osorio-Ramos, Eduardo Morales[17] used a feature for DTW and HMM similar to ours, which is the time-series data from the gestures. DTW performs the worst, with 82.75 percent accuracy. The relatively lower accuracy is caused by the fact that DTW does not allow the query gesture to have "jitter" behaviors, because DTW assumes linear time relations between the query and target gestures.

The HMM performance is similar to VMO. The estimation of the number of hidden states in an HMM is found by exhaustively searching over a range of possible values, which in a sense is similar to finding the optimal θ value for

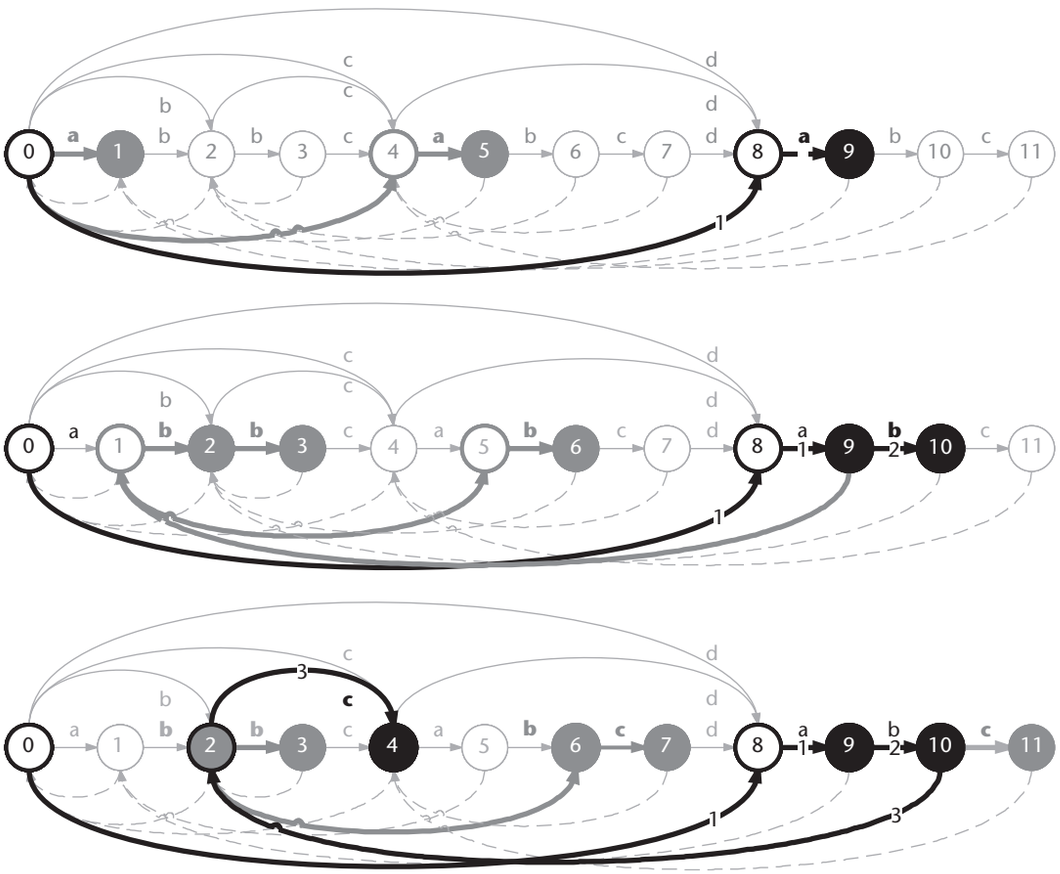

Figure 10. Decoding steps: (a) initialization and then decoding of (b) t = *2 and (c)* t = *3. Consider the target time series represented as the same VMO from Figure 1. In each subplot, parts marked black with thick arrows indicate the path for the chosen state, dark gray circles with thick arrows represent possible paths, and filled circles represent the candidate states. Numbers on the thick black arrows indicate the steps. In this example, the query* ***R*** *is assumed to have three frames and the subplots demonstrate hypothetic steps for the path started with frames in* ***O*** *in the cluster labeled "a" (among four possible paths started via a, b, c, or d). Here, the visualization of the query time series is omitted and the path is chosen generically to demonstrate Algorithm 6.*

VMO, but the expectation maximization (EM) algorithm for estimating the HMM parameters must iterate over each gesture for several times until convergence, while the construction for VMO requires only one pass through each gesture, thus making HMM relatively inefficient compared to VMO.

In Table 3, the criteria for a correct match are stricter than the previous experiment; we consider a match to be correct only if both the gesture and the participant who performed the gesture are the same as the query. The results in Table 3 indicate that, given the simple frame-by-frame 3D skeletal joint data points, the query-matching algorithm can retrieve not only the same type of gesture, but also the gesture that was performed by the same participant who did the query gesture. This result implies a possible solution to the user identification problem in gaming consoles or mobile devices, which was not addressed for the MSRC-12 dataset in previous research.

Gesture Following

Based on the online gesture-following algorithms proposed earlier, it is possible to track where in the stored multivariate time series the new observation should be mapped to. Figure 13 provides a visualization of such gesture

Require: Target signal in *VMO*,
Oracle($\mathbf{Q} = q_1; q_2; \ldots; q_T$; $\mathbf{O} = \mathbf{O}[1]; \mathbf{O}[2]; \ldots; \mathbf{O}[T]$) and
first frame of incoming time series $\mathbf{R}[1]$

1. Get the number of clusters, M $\leftarrow |\mathbf{\Sigma}|$
2. Initialize cost vector C $\in \mathbb{R}^M$ and path vector P $\in \mathbb{R}^M$
3. **for** $m = 1 : M$ **do**
4. $P_m \leftarrow$ Find the state, t, in the mth list from Σ with the least distance, d_m, to $\mathbf{R}[1]$
5. $C_m \leftarrow d_m$
6. **end for**
7. **return** Pargmin (C), P, C

(a)

Require: Target signal in *VMO*,
Oracle($\mathbf{Q} = q_1; q_2; \ldots; q_T$; $\mathbf{O} = \mathbf{O}[1]; \mathbf{O}[2]; \ldots; \mathbf{O}[T]$),
path vector P, cost vector C from Algorithm 7 and
incoming time series $R[n]$ at time n.

1. **for** $m = 1 : M$ **do**
2. $P_{temp} \leftarrow P_m$
3. $P_m \leftarrow$ Find the state, t, in lists with labels corresponding to forward links from state P_{temp} with the least distance, d_m to $R[n]$
4. C_m += d_m
5. **end for**
6. **return** $P_{\text{argmin}(C)}$, P, C

(b)

Figure 11. Gesture following: (a) Algorithm 7 (initialization) and (b) Algorithm 8 (online update). In Algorithm 7,* M *possible paths are initialized, returning a path vector* P *and a cost vector* C*. In Algorithm 8,* $P_{\text{arg min (c)}}$ *is returned as the found index in target time series* O*, indicating the best match between the incoming time series* R *and the stored time series* O*.

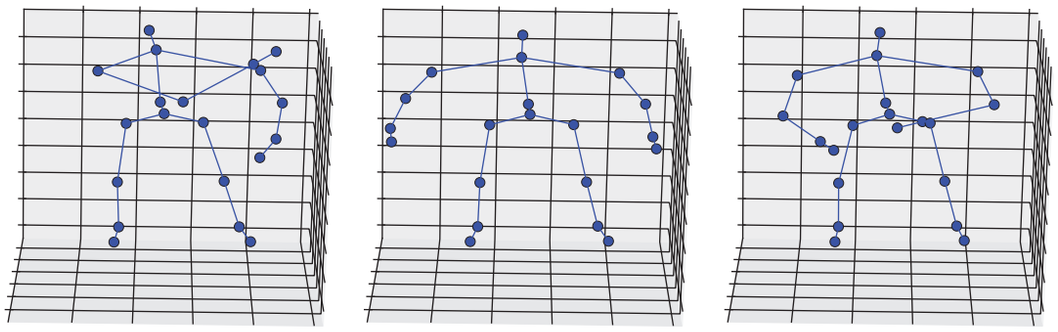

Figure 12. Example frames of 3D skeletal joints. The length of each gesture sequence is between 13 and 492 frames, and each participant performed 8 to 20 times for each gesture.

following. Figure 13a shows the result of the gesture following (red dashed line) based on a stored gesture (blue line) given unknown input observations (green dotted line). Figure 13b shows how the gesture-following result (red dashed line) corresponds to its original sequence (blue line) in terms of the matching in time (as of frame indices). The original

Table 1. Number of instances of the subset from MSRC-12 Kinect gesture dataset.

	Participant					
Gesture	**A**	**B**	**C**	**D**	**E**	**Total number**
Crouch or hide	20	20	21	21	20	102
Shoot with a pistol	20	10	22	9	20	81
Throw an object	9	10	10	20	22	71
Change weapon	19	20	20	20	20	99
Kick to attack	24	10	20	9	20	83
Put on goggles	10	11	12	10	20	63

Table 2. Precision for Kinect skeletal gesture retrieval considering only gesture type.

Approach	**Gesture**	**Precision (%)**
Variable Markov Oracle (VMO)	Crouch or hide	99.9
	Shoot with a pistol	90.6
	Throw an object	84.6
	Change weapons	98.4
	Kick to attack	92.7
	Put on goggles	92.2
	Avg. ± std.	93.1 ± 5
Dynamic time warping (DTW)[17]	Avg.	82.74
Hidden Markov Model (HMM)[17]	Avg.	91.81
SVM with Cov3DJ (support vector machine with covariances between joints as features)[16]	Avg.	93.6

Table 3. Precision for Kinect skeletal gesture retrieval considering both type and participant.

	Participant					
Gesture	**A**	**B**	**C**	**D**	**E**	**Total number**
Crouch or hide	99	89.5	87.1	86.2	98.5	92 ± 5.5
Shoot with a pistol	100	85	100	33.3	99.5	83.5 ± 25
Throw an object	63.3	83	81	94.5	86.8	81.7 ± 10
Change weapon	100	98.5	100	88.5	100	97.4 ± 4.4
Kick to attack	100	90	100	51.1	95	87.2 ± 18
Put on goggles	90	81.8	100	57	100	85.7 ± 15
Avg. ± std.	92 ± 13	87.9 ± 5.6	94.6 ± 7.7	68.4 ± 22.5	96.6 ± 4.7	Avg. ± std.

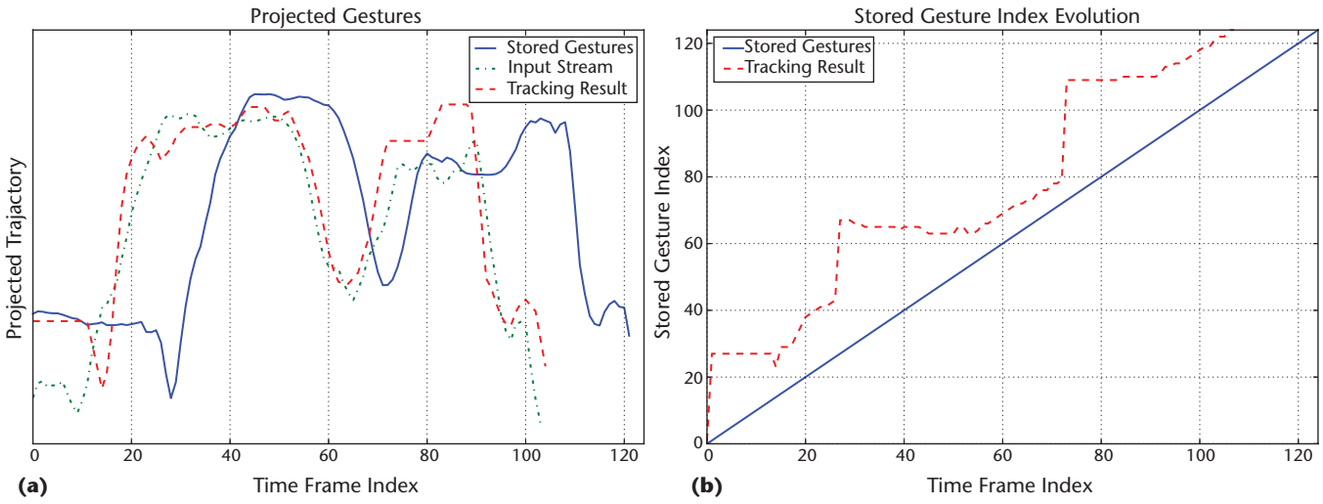

Figure 13. A visualization of gesture following: (a) projected trajectory of the stored gesture (blue), input stream (green), and gesture following result (red). (b) Time indices evolution of the gesture following result (red) compared to original gesture (blue).

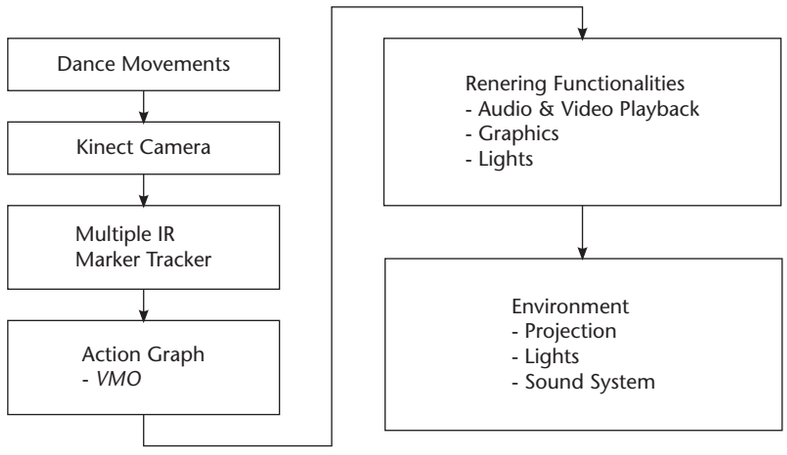

Figure 14. Diagram of the proposed interactive dance/graphics system using VMO. We use VMO as the mapping interface between the input dance movements and graphics/interaction/effects rendering during a performance.

sequence shown in the figure is part of a longer sequence, in which multiple gestures sampled from the MSRC-12 dataset were concatenated into one longer sequence to resemble how it would be used in a real world performance or gaming environment. This long time series is then projected onto its first principal component for visualization.

The example in Figure 13 shows that the online gesture-following algorithm can find the correct segment in the stored gesture, matching the input observations. For visualization purposes, we omit the rest of the stored gesture (blue line) where the other types of gestures are stored. Also, we notice from time index evolution in Figure 13b that the gesture following is consistent with the temporal relations of the stored gesture (the red dashed line mostly follows the diagonal with time stretch or compression movements indicated by the nearly vertical or horizontal lines).

We combine the aforementioned online gesture-following algorithms into an interactive dance/graphics system depicted in Figure 14. We use VMO as the mapping interface between the input dance movements and graphics/interaction/effects rendering during a performance. We implement VMO and the proposed algorithms in C++ within the openFrameworks open source environment (http://openframeworks.cc). Snapshots of VMO gesture following working alongside computer-generated graphics in openFrameworks are depicted in Figure 15. The dance movements are captured by identifying the infrared reflector tied on the dancer's joints using the Kinect camera.

We show an example of the raw depth image along with the identified marker in Figure 16. In Figure 15, the trajectory represents the stored gesture (sequence of 2D points) with different colors indicating mappings to different computer-generated graphics. The bigger circle is the input marker and the green small circle along the trajectory is the current gesture-following position on the stored gesture corresponding to the input marker. In this demo, the system generates different graphics when the

gesture following position (green circle) traverses onto different segments of the stored gesture, represented by different colors.

For the system proposed in this article, we consider a scenario in which the rehearsal or practice of a dancer is recorded and stored as VMO. During a live performance the system tracks the input dance movements and matches them to the gestures stored in VMO via Algorithms 7 and 8. For graphics/effects/interaction rendering, the temporal mapping between the live dance movements and the stored gestures lets us evolve the incoming time series by *scrubbing* another time series (such as a video) that varies in time. Other details of this interactive dance/graphics system are documented elsewhere.[18]

We envision extending our current progress in the study of VMO in both theoretical and practical directions. For the theoretical aspect, we plan to explore the comparison between VMO and HMM in terms of pattern-matching-based versus latent-model-based sequential modeling. In comparison to HMM, VMO relaxes the first-order Markov model limitation of HMM to include the variable context by having suffix links, but it loses generality in the absence of explicit emission probability for actual observations given state labels. Because the VMO estimation process includes selecting a threshold to determine the optimal model structure, we expect that frames with the same label in VMO will have implicitly probabilistic similarities as determined by the metric we use. We envision discovering such underlying probabilistic structures by relaxing HMM to its nonparametric relatives, as in Hierarchical Dirichlet Process HMM.[19]

We are also interested in exploring the possibility of exploring how the VMO construction algorithm might induce a metric using online clustering. The possible induced metric should help reveal the dynamic distribution embedded in the VMO data structure.

For practical extensions, such as dealing with the big data time-series problem, we are investigating implementations of VMO in databases (SQL, NoSQL, and so on) with efficient query functionalities.[20] Another line of research is the control improvisation problem,[21,22] in which human users can create new multimedia content by either specifying navigation rules for stored content using metarules or recombining stored media content using an input from other content. **MM**

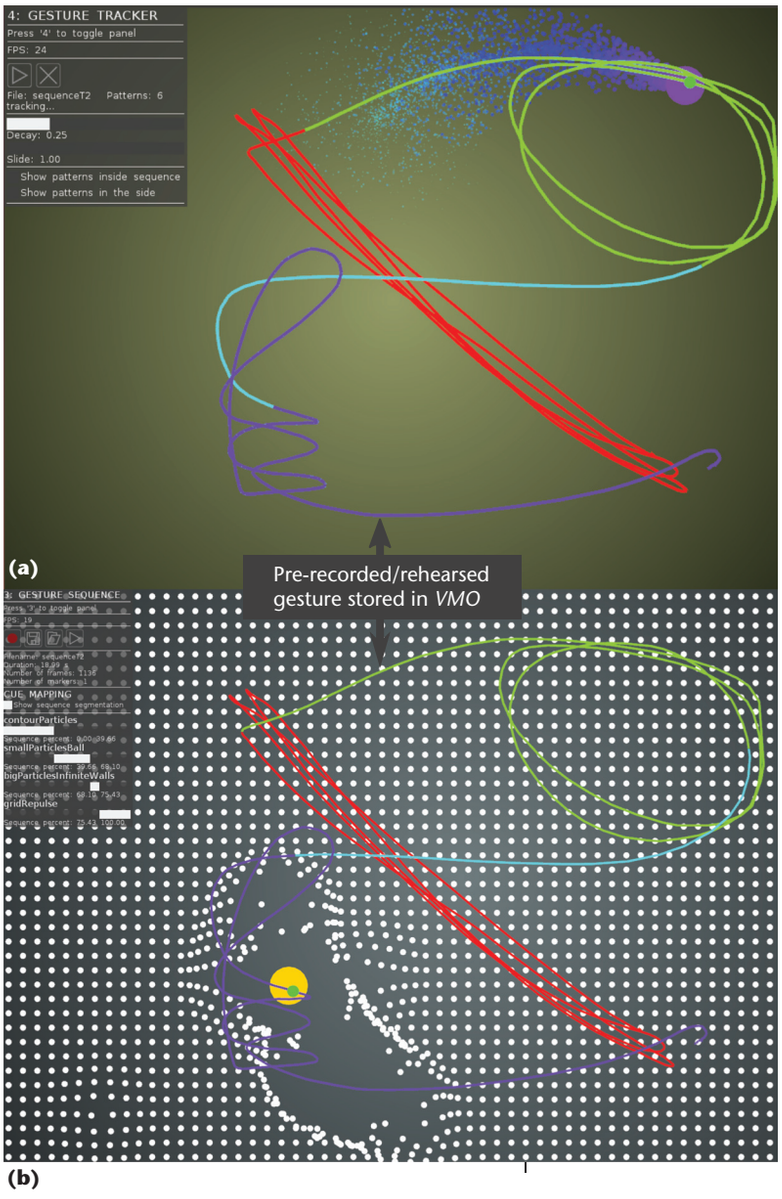

Figure 15. Snapshots of the openFrameworks application with embedded VMO and gesture-following algorithms: (a) A particle emission showing the trace of the input marker along the stored gesture trajectory; (b) grid particles interact with the input marker.

Acknowledgments

Cheng-i Wang is funded by the Center for Research in Entertainment and Learning (CREL) at Qualcomm Institute through the Office of Graduate Studies, University of

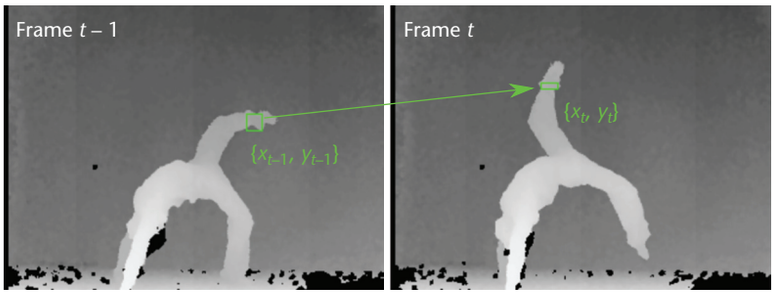

Figure 16. The green rectangles represent input identified markers on the dancer's ankle by processing the infrared images from the Kinect camera.

California, San Diego. We thank Tammuz Dubnov and Fabià Serra Arrizabalaga for technical support in developing the software needed to fulfill this research project.

This special issue is a collaboration between the 2014 IEEE International Symposium on Multimedia (ISM 2014) and *IEEE MultiMedia*. This article is an extended version of "Variable Markov Oracle: A Novel Sequential Data Points Clustering Algorithm with Application to 3D Gesture Query-Matching," presented at ISM 2014.

References

1. P. Esling and C. Agon, "Time-Series Data Mining," *ACM Computing Surveys*, vol. 45, no. 1, 2012, p. 12.
2. E. Unal et al., "Challenging Uncertainty in Query by Humming Systems: A Fingerprinting Approach," *IEEE Trans. Audio, Speech, and Language Processing*, vol. 16, no. 2, 2008, pp. 359–371.
3. A. Cont et al., "Guidage: A Fast Audio Query Guided Assemblage," *Proc. Int'l Computer Music Conf.*, 2007; https://hal.inria.fr/hal-00839071.
4. S. Dubnov et al., "Audio Oracle: A New Algorithm for Fast Learning of Audio Structures," *Proc. Int'l Computer Music Conf.*, 2007; https://hal.inria.fr/hal-00839072/file/AudioOracle_5.pdf.
5. C.-i. Wang and S. Dubnov, "Variable Markov Oracle: A Novel Sequential Data Points Clustering Algorithm with Application to 3D Gesture Query-Matching," *Proc. IEEE Int'l Symp. Multimedia* (ISM), 2014, pp. 215–222.
6. A. Lefebvre and T. Lecroq, "Compror: On-Line Lossless Data Compression with a Factor Oracle," *Information Processing Letters*, vol. 83, no. 1, 2002, pp. 1–6.
7. F. Bevilacqua et al., "Continuous Realtime Gesture Following and Recognition," *Gesture in Embodied Communication and Human-Computer Interaction*, Springer, 2010, pp. 73–84.
8. C. Allauzen, M. Crochemore, and M. Raffinot, "Factor Oracle: A New Structure for Pattern Matching," *SOFSEM 99: Theory and Practice of Informatics*, Springer, 1999, pp. 295–310.
9. G. Assayag and S. Dubnov, "Using Factor Oracles for Machine Improvisation," *Soft Computing*, vol. 8, no. 9, 2004, pp. 604–610.
10. A. Lefebvre, T. Lecroq, and J. Alexandre, "An Improved Algorithm for Finding Longest Repeats with a Modified Factor Oracle," *J. Automata, Languages and Combinatorics*, vol. 8, no. 4, 2003, pp. 647–657.
11. R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification*, John Wiley & Sons, 2012.
12. F. Pachet and P. Roy, "Markov Constraints: Steerable Generation of Markov Sequences," *Constraints*, vol. 16, no. 2, 2011, pp. 148–172.
13. S. Dubnov, G. Assayag, and A. Cont, "Audio Oracle Analysis of Musical Information Rate," *Proc. Fifth IEEE Int'l Conf. Semantic Computing* (ICSC), 2011, pp. 567–571.
14. M.J. Wainwright and M.I. Jordan, "Graphical Models, Exponential Families, and Variational Inference," *Foundations and Trends in Machine Learning*, vol. 1, nos. 1-2, 2008, pp. 1–305.
15. S. Fothergill et al., "Instructing People for Training Gestural Interactive Systems," *Proc. SIGCHI Conf. Human Factors in Computing Systems*, 2012, pp. 1737–1746.
16. M.E. Hussein et al., "Human Action Recognition Using a Temporal Hierarchy of Covariance Descriptors on 3D Joint Locations," *Proc. 23rd Int'l Joint Conf. Artificial Intelligence*, 2013, pp. 2466–2472.
17. M. Ramírez-Corona, M. Osorio-Ramos, and E.F. Morales, "A Non-Temporal Approach for Gesture Recognition Using Microsoft Kinect," *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, Springer, 2013, pp. 318–325.
18. T. Dubnov, Z. Seldess, and S. Dubnov, "Interactive Projection for Aerial Dance Using Depth Sensing Camera," *IS&T/SPIE Electronic Imaging*, Int'l Soc. Optics and Photonics, 2014; doi: .
19. Y.W. The et al., "Hierarchical Dirichlet Processes," *J. American Statistical Assoc.*, vol. 101, no. 476, 2006, pp. 1566–1581.
20. A. Charapko and C.-H. Chuan, "Indexing and Retrieving Continuations in Musical Time Series Fata Using Relational Databases," *Proc. IEEE Int'l Symp. Multimedia* (ISM), 2014, pp. 341–346.

21. A. Donzé et al. "Machine Improvisation with Formal Specifications," *Proc. 40th Int'l Computer Music Conf.* (ICMC), 2014, pp. 1277–1284.
22. C.-i. Wang and S. Dubnov, "Guided Music Synthesis with Variable Markov Oracle," *Proc. 3rd Int'l Workshop on Musical Metacreation, 10th Artificial Intelligence and Interactive Digital Entertainment Conf.*, 2014.

Cheng-i Wang is a PhD student in the Department of Music at the University of California, San Diego. His research interests include machine improvisation and music information retrieval. Wang received his Master of Music in music technology from New York University. Contact him at chw160@ucsd.edu.

Shlomo Dubnov is a professor in the Department of Music at the University of California, San Diego. He holds a doctorate in computer science from the Hebrew University. He is a senior member of IEEE and serves as a Director of the Center for Research in Entertainment and Learning at UCSD's Qualcomm Institute CALIT2. Contact him at sdubnov@ucsd.edu.

cn *Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.*